Artifact or Process Guidance, an Empirical Study

Marcos Aurélio Almeida da Silva[1⋆], Alix Mougenot[1⋆⋆], Reda Bendraou[1], Jacques Robin[1] and Xavier Blanc[1]

LIP6, UPMC Paris Universitas, France

Abstract. CASE tools provide artifact guidance and process guidance to enhance model quality and reduce their development time. These two types of guidance seem complementary since artifact guidance supports defect detection after each iterative development step, while process guidance supports defect prevention during each such step. But can this intuition be empirically confirmed? We investigated this question by observing developers refactoring a UML model. This study attempted to assess how general were the observations made by Cass and Osterweil on the benefits of guidance to build such model from scratch. It turns out that they do not generalize well: while their experiment observed a benefit on quality and speed with process guidance (but none with artefact guidance), we, in contrast, observed a benefit on quality at the expense of speed with artefact guidance (but none with process guidance).

1 Introduction

CASE tools provide automated services that assist a development team to perform iterative artefact construction and revision more systematically, safely and efficiently. Two key classes of such services are *artifact guidance* and *process guidance*. The former assists developers to produce artifacts that conform to some structural pattern, while the latter assists them in following a step-by-step plan whose execution results in such pattern-conforming set of artefacts [**?**,**?**,**?**].

One key subclass of artifact guidance service is structural pattern violation detection and diagnosis [**?**,**?**,**?**][1]. One key subclass of process guidance consists of plan deviation detection and diagnosis [**?**]. Intuitively, these respective subclasses of artefact guidance and process guidance seem to bring complementary software quality gains, since the former focuses on defect *detection after* each iterative development step, while the latter focuses on defect *prevention during* each such step. But can this intuition be empirically confirmed? Do the two together provide a quantitatively significative improvement over each of them alone (and hopefully over no guidance whatsoever)? Beyond software quality,

⋆ This work was partly funded by ANR project MOVIDA Convention N° 2008 SEGI 011.

⋆⋆ This work was partly funded by the french DGA.

[1] Another one is violation repair suggestion [**?**,**?**]

what are their respective and combined effects on development speed? Is the time overhead involved in performing each of them alone or together clearly inferior to the time they allow saving through artefact defect prevention and quick correction?

Cass and Osterweil [**?**] pioneered the investigation of these crucial questions through a first experiment in which 16 novice designers were asked to build from scratch a UML class diagram that conformed to the MVC architectural pattern[**?**]. The single quality metric used in this experiment was the degree of conformance of the diagram to this pattern, as graded by a group of specialists; and the single efficiency metric was the time spent building the diagram. These parameters were measured for four subject subgroups that respectively used no guidance, artifact guidance alone, process guidance alone and both guidance. This experiment yielded one expected result, that following process guidance alone significantly improved both the quality and efficiency metrics, and one unexpected result, that no significative effect was measured for artifact guidance on these metrics.

In this paper, we report on an experiment that furthers the Cass and Osterweil's research by trying to generalize their results. In order to be able to draw clear insights from our experiment, we designed it to differ from Cass and Osterweil's by a minimum two controlled features. The first is the starting point of the experiment. We asked our subjects to *refactor* a given UML diagram that did not conform to the MVC pattern into one that does, instead of *building* a conform diagram *from scratch*. The second differing feature of our experiment is the process guidance that we provided to the subjects. This was an unavoidable consequence of the first, since a refactor process obviously differs from a construction one.

We kept the other features of our experiment, such as the dependent and independent variables and the subject partitioning method, as similar as possible to those of the Cass-Osterweil experiment. While their experiment reported significant benefits for process guidance but no effect for artifact guidance, ours intriguingly reports completely different results: regarding artifact guidance, significant negative effect in terms of efficiency but no significant effect in terms of quality; regarding process guidance, no significant effect at all. It suggests that a lot more experimental research is needed before we can clearly determine which guidance is more beneficial under which circumstances.

For our experiment, we followed the empirical experiment protocol defined in [**?**]. The rest of the paper follows the structure advocated by this protocol: Section 2 presents the definition and planning of the empirical study. Section 3 presents its instrumentation. Section 4 presents our concrete realization of this study and synthesizes our results. Finally, Section 5 we discuss our results before concluding in Section 6.

2 Experiment definition and planning

In this section, we define the experiment following Wohlin's approach [**?**]. Section 2.1 briefly defines the goal of our experiment. Section 2.2 presents the decisions we took when planning this experiment. Then section 2.3 presents the experiment null hypothesis. Finally, section 2.4 identifies potential threats to its validity.

2.1 Experiment overall definition

According to [**?**], an empirical study is essentially defined by five features: object(s), purpose, quality focus, perspective and context. The essence of any experiment can thus be summarized by a single sentence that briefly defines all five of these features. Definition 1 summarizes our experiment this way.

Definition 1 (Goal of the empirical study). *Analyze the* ***refactoring of a UML class diagram to turn it conformant to the MVC pattern*** *for the purpose of* ***artifact and process guidance evaluation*** *with respect to* ***efficiency and quality*** *from the perspective of a* ***software architect and quality manager*** *in the context of* ***novice designers*** *using a* ***UML CASE tool****.*

Each experiment feature is briefly elaborated below:

Object: the object of this study consisted in refactoring a UML class diagram to make it follow the Model-View-Controller pattern[**?**].

Purpose: The general purpose of this empirical study was to measure whether or not process and artifact guidance do provide efficiency and quality benefits for developers.

Quality Focus: This study considered that benefits provided by process and artifact guidance were efficiency and quality. Efficiency was measured as the time needed to complete the object. Quality was measured as the degree of conformance to the MVC design pattern.

Perspective: The intent of this study was to be interpreted by a software architect and quality manager. These professionals have the skills necessary to define both kinds of guidance and to evaluate the result of their respective use.

Context: The subjects were master students, at the end of their course. To refactor a UML class diagram into the MVC pattern, they used the UML CASE tool Papyrus [**?**], extended by three plug-ins that we implemented specially for the experiment.

Our experiment had a single object carried out by multiple subjects. It can thus be classified as a **muti-test within object** experiment in the taxonomy of empirical studies [**?**].

2.2 Experiment planning

Object As there are many flavors of MVC patterns, we used the one popularized by Apple [**?**]. It requires the classes and interfaces of the model to be partitioned into three components: the Model, the View and the Controller. The model classes hold the persistent data structure of the system, the view classes hold the visual elements of the GUI and the controller classes handle the events generated by user interaction with the view classes into calls to business code in the model. The pattern prohibits any direct coupling between model and view classes. All interactions between them must be mediated through controller classes that together also encapsulate the overall system's control flow.

We decided to take the input class diagrams to refactor from an implemented running application used as case study. This application is a toy maze game composed of 16 classes. We chose this particular example because, on the one hand, refactoring a significantly smaller system would be too simple. On the other hand, a significant bigger system would make the input diagram to suffer from another design defect, namely lack of cognitively manageable substructures, that is orthogonal to the focus of our experiment, namely non-conformance to an architectural pattern. It would thus have muddled the interpretation of our results. The input class diagrams violated the MVC patterns in various ways. It contained classes that grouped together features related to view and model aspects. It also contained direct dependencies between classes that only contained features related to view aspects and classes that only contained features related to model aspects[2].

Purpose We decided to specify the artefact guidance thanks to a set of inconsistency rules[**?**]. These rules (see Figure 1) enforce: (i) the presence of the correct packages in the expected pattern (rules 1–6); (ii) the fact that the classes in the model (respectively view) should not depend on classes outside of it (rules 7 and 8); (iii) the fact that graphical classes[3] should be in the view package and that every class should be in one of the three packages (rules 9 and 10).

We decided to specify the process guidance as a software process that subjects were asked to follow. Figure 2 represents this process. It consists of 6 tasks whose objective is (i) create the model, view and controller packages (task 1); (ii) move the classes that trivially belong to one of the packages to it (tasks 2–4) and then (iii) breaking the responsibilities between the classes that would belong to more than one package and then move the remaining classes to the controller (tasks 5–6).

Notice that this process neither intends to be a general solution for this problem, nor will produce the most elegant solutions for it.

[2] The complete set of artifacts used during this experimentation is available at *http://meta.lip6.fr/?page_id=186*.

[3] To facilitate the identification of a graphical class by our subjects, who were all Java programmers, all these class in the diagram to refactor extended a class from one of Java's graphical API.

1. The model should own exactly three packages.
2. Nesting packages is forbidden.
3. The Model package is missing.
4. The View package is missing.
5. The Controller package is missing.
6. Empty packages are forbidden.
7. Some class in the Model package references another class outside of this package. Only classes in the Controller package can reference classes outside of it.
8. Some class in the View package references another class outside of this package. Only classes in the Controller package can reference classes outside of it.
9. Some class is graphical and is not in the View Package.
10. Some class is not included in any package. All classes must be included in either the Model, View or Controller.

Fig. 1. Artifact guidance

1. **createMVCPackages:** for this task, create three packages called *Model*, *View* and *Controller*;
2. **movePureModelClassesToModel:** for this task, move the classes with pure model responsibilities (i.e., that represent persistent data and business rules) to the model package;
3. **movePureViewClassesToView:** for this task, move the classes with pure view responsibilities (i.e., that only contain GUI features) to the view package;
4. **movePureControllerClassesToController:** for this task, move the classes with pure controller responsibilities (i.e., that define the control and data flow between model and view classes) into the controller package;
5. **extractMixedViewClasses:** for this task, divide classes with mixed view and controller responsibilities into one class with pure view responsibilities and one class with pure controller responsibilities and link them an association (from the controller one to the view one);
6. **moveAllRemainingClassesToController:** for this task, move the remaining classes to the controller package;

Fig. 2. Process Guidance

Quality Focus We decided to measure the effects of the artefact and process guidance with respect to design time and quality. Where "design time" is a ratio variable measuring the time in minutes needed for a subject to carry out the refactoring. Regarding quality, we decided to measure the quality of the final models as a ratio measure, by giving 20 quality points to each model and then decreasing this number by a fixed number of points for each common mistake (see Table 1[4]).

Mistake	**Points**
Minor dependencies not broken	-1
Forbidden dependencies partially handled	-2
Operations or parameters missing in the final model	-2
Ignored forbidden dependencies	-5
Graphical classes in controller	-6
Model classes in controller	-7
Most controller classes not in controller	-7
Out of scope	-20

Table 1. The most common mistakes found in the students' models

Our empirical study then defines two independent variables (G_a for artifact guidance and G_p for process guidance) and two dependent variables (T for the time needed to carry out the task and Q for the quality of the result). The experiment thus follows a two factors with two treatments design. Table 2 presents the four possibilities. Each of these possibilities represents a treatment of our study.

	$G_a = false$	$G_a = true$
$G_p = false$	A = no guidance	B = artifact guidance
$G_p = true$	C = process guidance	D = both

Table 2. Two factors with two treatments experiment

Context We decided to choose software engineering master students at the end of their cursus as our subjects. This choice was motivated by two factors: (1) the low-cost availability of such subjects and (2) their level of experience in designing that is comparable to the subjects of the Cass-Osterweil experiment (novice).

[4] We defined "minor dependencies" as the ones generated by types of parameters of operations that are harder to find on the model. Additionally, we defined"forbidden dependencies" as dependencies between the MVC packages that are not allowed by the pattern, e.g. a dependency from a Model class to a view class.

Regarding the subject set partitioning; our experiment used a stratified random sampling. The 21 subjects of our experiment were partitioned into three skill levels corresponding to their average grade in the MSc. program. Random sampling was then applied over this skill partition to create treatment partitions of equal average skill. One group with six students and three groups with five were created and randomly assigned to our four treatments as represented in Table 2.

2.3 Hypothesis

The following hypotheses are specified to evaluate the impact of artifact and process guidance on design time and quality. These four hypotheses specify that the use of artifact guidance and process guidance do not provide any improvement to the dependent variables. If any of these hypotheses is rejected, the corresponding guidance has a significant effect. Deciding whether such effect is positive or negative is done by comparing the averages of the values of the variables in each data set.

Definition 2 (Null Hypothesis).

$H_0(1)$ *Design time without process guidance is equal to design time with process guidance.*

$H_0(2)$ *Design time without artifact guidance is equal to design time with artifact guidance.*

$H_0(3)$ *Design quality without process guidance is equal to design quality with process guidance.*

$H_0(4)$ *Design quality without artifact guidance is equal to design quality with artifact guidance.*

2.4 Threats to Validity

Internal validity. We avoided selection and history bias by stratifying the subjects by skills and then assign them to group of equal average skill. We avoided instrument threats by having all subjects using the same tool (modulo the extra-views defining the independent guidance variable differences).

External validity. The results we obtained with masters students of our university may not be generalizable to other groups of novice designers and even less to expert designers. The specificity of the input class diagram to refactor, as well as the artifact and process guidance services that we proposed might also represent a threat to the external validity of our experiment.

Construct validity. Determining whether a class diagram follows or violates the MVC pattern ultimately involves cognitive judgment calls made by subjects based on their understanding of the contextual semantics of each class. This judgment is thus in part subjective and cannot be fully automated. It directly threatens the validity of the dependent variable Q. Since subjects deliver their refactored diagram when they either judge it fully conform to

the MVC pattern or when they no longer can find a way to improve its level of conformity, this partial subjectivity also indirectly threatens the validity of the dependent variable T.

3 Instrumentation

Our experiment's instrumentation consisted of an UML CASE tool built on top of Eclipse. It provides four perspectives, one for each treatment. All those perspectives include four basic views provided by the open-source Papyrus UML editor[5] (see Section 3.1). The perspectives for the artefact guidance treatments (B and D) add the Artefact View (see Section 3.2), while the perspective for the process guidance treatments (C and D) add the Process View (see Section 3.3).

3.1 Basic Views

Figure 3 shows the perspective with no guidance (treatment A). This perspective is composed of the following four basic views :

1. **Editor Area View:** this view contains the UML class diagram to be refactored. This diagram is displayed and edited with the Papyrus UML2 editor.
2. **Property Pane View:** this view allows the subjects to manipulate the properties of the elements in the diagram.
3. **Outline View:** this view displays the model's element tree. It provides another representation of the model that complements the class diagram.
4. **Birdview:** this view displays an overview of the whole class diagram zoomed out.

3.2 Artifact Guidance View

Figure 4 shows the Artifact Guidance View. The Artifact Guidance View shows the number of inconsistencies found in the model (1) and the list of model elements that violate each design rule (2).

3.3 Process Guidance View

The Process Guidance View is based on the Eclipse Cheatsheets [**?**] as shown in Figure 5. It displays the process by the means of a tree of tasks (see area numbered (1) in fig. 5). When executing a task, this view shows a detailed textual description of it (see area numbered (2) in fig. 5), which explains the concepts underlying the actions to perform and the expected state of the model at the end of this task. Finally, at the bottom of the screen, a link reading "Click here to finish" is provided (see area numbered (3) in fig. 5). A click on it causes the model state to be validated against the task's postcondition (e.g., at

[5] *http://www.papyrusuml.org/*

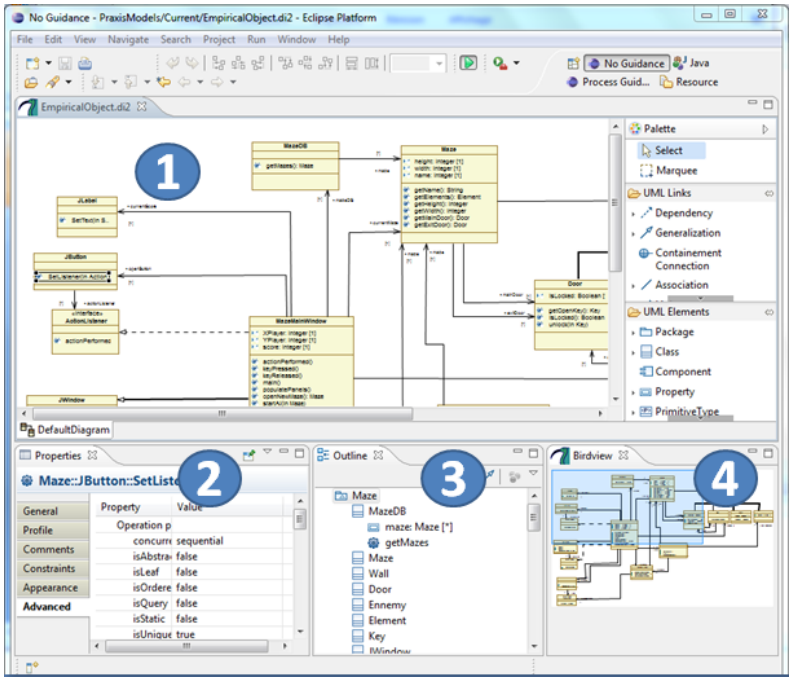

Fig. 3. Basic Environment

the end of a task "Create X" the model is validated to make sure "X" exists). A task postcondition can be seen as an finer grained form of artifact guidance for intermediary model states). If it is violated, the tool gives the subject two options: (i) going back to the task to correct the model or (ii) disregard the tool's warning and move on to the next tasks in the process.

4 Operation and results

4.1 Execution of the experiment

The experiment described in the last section was carried out by 21 subjects over 2 consecutive hours. These two hours where divided in four periods: three training periods lasting 10 minutes each, following by one 90 minutes period to complete the task.

The first two training period consisting in viewing video tutorials: the first on the Papyrus UML CASE tool and the second on the artifact and process guidance plug-ins (each group saw only the parts of the videos relevant to its assigned treatment). These videos remained available as a form of help, on each machine while the subject refactored the input model. During the third training period, the subjects were asked to experiment using with the CASE tool.

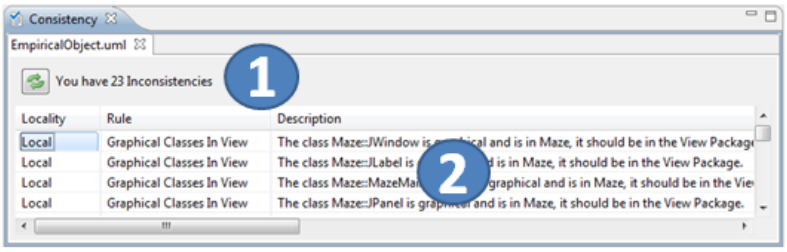

Fig. 4. Artifact Guidance View

After this 30 min of training, the subjects they received a textual description of the MVC pattern together with the input model to refactor. As soon as the last subject received this description, they started to work on it. The subjects had autonomy to decide when they felt that they had completed refactoring. The CASE tool stored the model that they submitted as refactored, together with the log of the editing commands they executed to obtain it. The final model was reviewed to assign it a quality measure. The log could be used to perform post-mortem finer grained analysis. The impact of the subject's degree of adherence to the process guidance on both speed and final model quality was beyond the scope of this study.

4.2 Results and Analysis

Figure 6 shows the results obtained by the subjects regarding the time needed to perform the refactoring (T) and the quality of the returned design (Q). A n/a in the time column indicates that the subject has not finished the refactoring in the time frame.

We used the ANOVA[6] method against these results to asses the effect of the guidance. This tests assumes normal distribution and interval scale data. We confirmed the normality of our data set by executing the Shapiro-Wilk normality test. We obtained respectively $\rho = 0.01055$ and $\rho = 0.04667$ for T and Q. In the case of the subjects that had not completed the refactoring ($T = n/a$), only the design quality of the produced models was considered.

The following observations were made:

- The $H_0(1)$ null hypothesis has **neither** been rejected **nor** accepted with $\rho = 0.602$. As a consequence, the difference in the average design **time** of the groups with **process** guidance (groups C and D, average design time = 50.8 min) and the groups without it (groups A and B, average design time = 58.7 min) is **not** statistically significant.

[6] This test is used to compare the variances of two data sets in order to determine if they present statistically significative differences. The lower the obtained ρ-value the more significative is the difference.

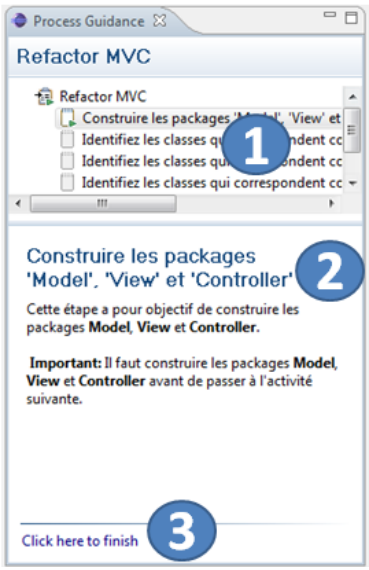

Fig. 5. Process Guidance View

– The $H_0(2)$ null hypothesis has been **rejected** with $\rho = 0.001$. As a consequence, the difference in the average design **time** of the groups with **artifact** guidance (groups B and D, average design time = 62.9 min) and the groups without it (groups A and C, average design time = 49.3min) **is** statistically significant.
– The $H_0(3)$ null hypothesis has **neither** been rejected **nor** accepted with $\rho = 0.488$. As a consequence, the fact that the average design **quality** of the groups with **process** guidance (groups C and D, average design quality = 15.2) and the groups without it (groups A and B, average design time = 15.2) is the same is **not** statistically significant.

A (no guidance)		B (artefact guidance)		C (process guidance)		D (both guidances)	
$T(min)$	Q	$T(min)$	Q	$T(min)$	Q	$T(min)$	Q
47	8	63	20	47	15	59	20
34	15	63	19	40	13	n/a	15
66	13	65	15	32	15	67	14
68	14	63	18	n/a	4	60	12
60	15	n/a	10	n/a	0	n/a	9
n/a	0						

Fig. 6. Time (T) and quality (Q) results for each treatment.

– The $H_0(4)$ null hypothesis has neither been rejected nor accepted with $\rho = 0.523$. As a consequence, the difference in the average design quality of the groups with artifact guidance (groups B and D, average design quality = 16.7) and the groups without it (groups A and C, average design quality = 13.6) is not statistically significant. However, upon finer analysis, when considering one subject of group A as an outlier, due to the fact that the quality of result s/he delivered is far off the average of other subjects, then the $H_0(4)$ null hypothesis is **rejected** with $\rho = 0.006$ and the difference becomes statistically significant.

5 Result interpretation

5.1 Result synthesis

The results we measured and that have been validated thanks to the ANOVA method can be synthesized by the Definitions 3 and 4:

Definition 3 (Significant effects).

Subjects that used the artefact guidance took more time to perform the refactoring, but delivered design of better quality.

Definition 4 (Non significant effects).

No significant effects were measured for the design quality and time with process guidance.

5.2 Discussion

Beyond the overall statistical results given above, we made a set of interesting finer grained observations. Subjects provided with artefact guidance considered their refactoring completed only when the error list displayed was empty. This suggests that such guidance helps developers decide when they have completed a refactoring task.

The initial model contained 5 initial errors. Therefore, at the beginning of the study, five errors were displayed in the error list. During the study, we observed that up to 20 errors could be detected by the artefact guidance and displayed in the error list. Those errors obviously didn't have the same complexity to solve. At the end of the study, all of the subjects provided with artefact guidance were impeded by a same error. This error targeted one of the UML class that had to be split in two parts as it has a mixed responsibilities. Subjects spent much time on resolving this error that were, indeed, more complex to correct. This suggests that developers with artefact guidance took much more time than the others as they all try to solve all the errors, including the difficult ones.

Subjects provided with process guidance complete process 1 to 4 quite quickly, but they spend more time on tasks 5 and 6. This may be explained by the fact that tasks 1-4 are defined in very prescriptive terms, easy to understand and can

be carried out with a very short editing action sequence. This was also the case of the tasks in the Cass-Osterweil experiment that observed benefits to process guidance. In contrast, the definition of tasks 5-6 are more descriptive. Mapped them onto an editing action sequence require more design skills and a deeper understanding of the class contextual semantics. This may explain why no effect has been measured for the process guidance regarding efficiency and quality.

Since we designed our study to differ from Cass and Osterweil's [**?**] by a minimal two controlled features, the fact that we observed almost opposite results may suggest that the conclusion of both studies do not generalize well. As a consequence, we argue that the benefits of the artefact and process guidance seem to be not measurable in general. Rather, regarding a specific pair of artefact and process guidance, it should be possible to statistically measure their respective effects.

6 Conclusion

In this paper, we presented an empirical study that aimed at measuring the effects of two types of services provided by advanced CASE tools: artefact guidance and process guidance. More precisely, we measured the effects on the time required to perform a model refactoring and on the quality of the resulting model.

This is the first study of this kind, though a similar study was carried out for the task of building a model from scratch [**?**]. This earlier study reported that for such task, process guidance statistically significantly improved both design time and quality, while artefact guidance had no statistically significant impact. In contrast, for the present task, we report no statistically significant impact for process guidance. We also report that artefact guidance statistically significantly increases design time and that it only statistically significantly improves the quality of the resulting models if we discard one subject as an outlier.

Our study shows that much further research needs to be carried out in this field before we can answer, on strong statistical ground, to questions such as:

- For what tasks are artefact and process guidance complementary and for what tasks are they redundant?
- Are there tasks for which neither guidance positively impact on development speed and software quality?
- What subclasses of artefact and/or process guidance are more helpful than others?